¿Qué son?

Son pequeños programas que se pueden ejecutar de varias formas, entre ellas, de forma interactiva desde una terminal. Normalmente, los comandos suelen aceptar parámetros para extender su funcionalidad.
Gran parte de los servidores están basados en sistemas Unix, como Solaris, BSD, HP-UX, así como las distintas distribuciones de Linux que también son muy utilizadas en el ámbito personal, entre ellas Ubuntu, Debian, CentOS, Red Hat, etc. También MacOS está basado en Unix. El conocimiento de estos comandos es esencial para poder administrar y ser más productivo en estos sistemas.

¿Dónde se encuentran?

Al introducir un comando en una terminal, el intérprete de comandos lo busca en directorios incluidos en lo que se denomina PATH. El PATH es una variable de entorno que contiene una lista con los directorios del sistema más comunes para almacenar ejecutables, por ejemplo /bin o /usr/bin.

Para ver los directorios que contiene el PATH, podemos ejecutar el comando *echo $PATH*. Cada uno de ellos estará separado por dos puntos. Si se ejecuta un comando que el sistema no encuentra, es debido a que no está en ninguno de estos directorios.

¿Cómo saber qué hace un comando?

Para saber qué hace un comando, se puede usar la herramienta **man**. Esta herramienta mostrará la documentación del comando que se le pase. Este manual suele estar dividido en secciones, de las cuales la más importante es la primera, que contiene la información general del comando. Los comandos también suelen tener su propia ayuda ligera con –help o -h.

¿Cómo añadir un comando?

Existen varias formas de añadir un comando para que se pueda usar desde una terminal.

- **Gestor de paquetes:** a través de un gestor de paquetes como *Homebrew*, *Apt* o *Pacman,* se pueden instalar comandos de sus repositorios. Estos gestores se encargan de automatizar la instalación, la actualización o el borrado de los programas. Automáticamente se instala el ejecutable en una ruta perteneciente al PATH.

- **Programación de comandos:** ocasionalmente necesitaremos programar un script personalizado para realizar alguna tarea. Para poder ejecutarlo como un comando, será necesario moverlo a un directorio del PATH o incluir el directorio donde se encuentre en el PATH. Para ello, simplemente podemos utilizar la instrucción:

  export PATH=$PATH:/ruta

¿Para qué sirven?

En sistemas de tipo Unix, es bastante común usar la terminal para moverse de un lado a otro y realizar las acciones desde allí, en lugar de usar un explorador de archivos o cualquier otro programa con interfaz gráfica. Para ello existen los comandos **cd** o *change directory* y **pwd** o *print working directory*.

pwd

El comando **pwd** muestra el directorio actual de trabajo, es decir, dónde estamos situados y dónde se van a ejecutar los comandos posteriores que se introduzcan.

Mostrará la ruta absoluta del directorio, esto es, la ruta partiendo desde la raíz del sistema.

Este comando acepta solamente dos parámetros:

- **-L:** este parámetro se aplica por defecto, y lo que hace es mostrar la ruta lógica en la que nos encontramos. Esto quiere decir que si navegamos al directorio */var/run* nos mostrará esta misma ruta.

- **-P:** este parámetro muestra la ruta física en la que nos encontramos. Esto quiere decir que si nos encontramos en un directorio que apunta a otro mediante un enlace simbólico, con este parámetro veremos dónde estamos físicamente. Por ejemplo, ***pwd -P*** en /var/run muestra la salida /private/var/run.

cd

El comando **cd** cambia el directorio actual de trabajo al directorio que se le pase como argumento. Se le puede pasar tanto la ruta absoluta como la relativa, desde donde nos encontramos. Se utilizan dos puntos (..) para referirse al directorio padre y un punto (.) para referirse al directorio actual. Si no se le pasa ningún directorio a este comando, moverá el directorio actual de trabajo al directorio **home** del usuario (*/home/{usuario}*, ~).

Acepta cuatro parámetros de los cuales dos son los más importantes:

- **-L:** se aplica por defecto y mueve el directorio de trabajo a la ruta lógica indicada, por ejemplo *cd /var/run*.

- **-P:** mueve el directorio de trabajo a la ruta física correspondiente a un enlace simbólico. Por ejemplo, *cd -P /var/run* mueve a */private/var/run*.

Pila de directorios: pushd y popd

autentia

¿Qué es?

Mientras usamos una terminal, es muy común querer navegar entre dos directorios múltiples veces. Existe una herramienta llamada **pila de directorios**, que nos ayuda con esta tarea. En ella se pueden ir almacenando directorios para posteriormente movernos al directorio que está en la cima de la pila.a través de los comandos pushd y popd.

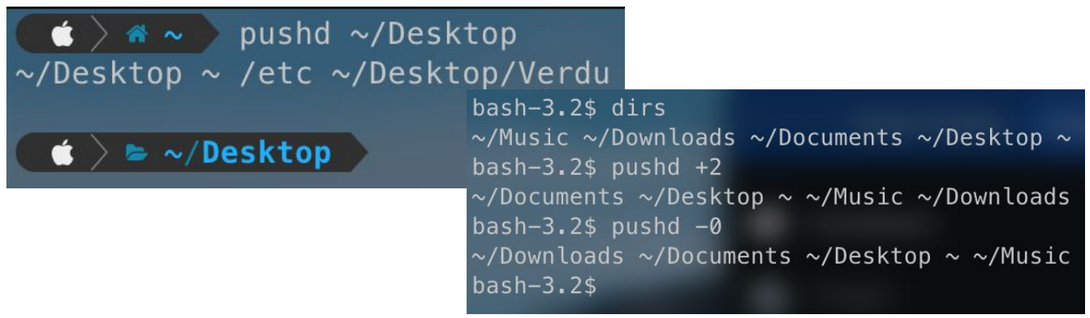
pushd

El comando **pushd** añade un directorio a la cima de la pila y mueve el directorio actual de trabajo al añadido a la pila. Si no se le pasan argumentos, intercambia los dos directorios superiores de la pila.

Para ver el contenido de la pila de directorios, se puede usar el comando **dirs**.

El comando **pushd** acepta un parámetro, **-n**, que permite añadir un directorio a la pila pero sin cambiar el directorio actual de trabajo, como haría el comportamiento normal del comando.

A parte del directorio, admite un argumento (+/-) N, que rota la pila para que el directorio número N de la pila empezando por la cima (+) o por el fondo (-) se encuentre en la posición superior, moviéndonos a ese directorio.

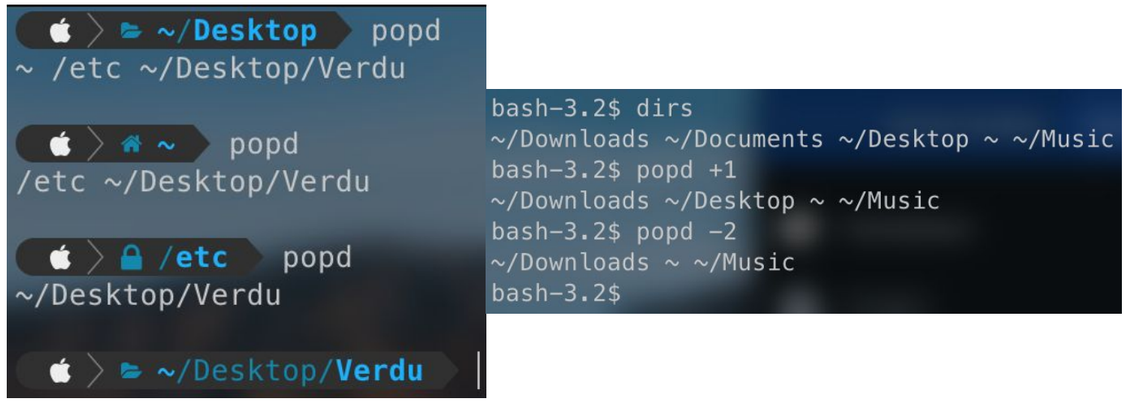


popd

El comando **popd** elimina el directorio de la cima de la pila y mueve el directorio actual de trabajo a este directorio.

El comando **popd** admite un parámetro, **-n**, que parecido a pushd, permite eliminar un directorio de la pila sin cambiar el directorio actual de trabajo.

A parte del directorio y al igual que pushd, admite un argumento (+/-) N para eliminar de la pila el directorio número N por la cima (+) o por el fondo (-). El directorio seleccionado se eliminará pero moverá allí el directorio actual de trabajo.



¿Para qué sirve?

El comando **ls** o *list* es de los comandos más típicos de sistemas Unix, cuando se trata de navegar entre directorios por terminal, ya que nos permite ver el contenido del directorio actual de trabajo.

¿Cómo funciona?

El comando **ls** muestra un listado con el nombre de los archivos y directorios que se encuentran en el directorio actual de trabajo. Funciona también si se le pasa la ruta de otro directorio como argumento.

Acepta muchos parámetros, según se quiera ver unos datos u otros :

- **-a:** muestra los archivos ocultos (empiezan por punto)
- **-F:** muestra caracteres después de los archivos para indicar a qué tipo pertenecen (por ejemplo / después de los directorios o * después de algún ejecutable)
- **-G:** activa colores en la salida (muchos intérpretes de comandos ya lo activan automáticamente)
- **-R:** lista recursivamente los subdirectorios encontrados
- **-S:** ordena por tamaño, de mayor a menor
- **-l:** lista los archivos en formato largo o detallado
- **-h:** junto con el -l, muestra sufijos para las unidades (Kb, Gb, etc)
- **-r:** invierte el orden de la salida
- **-t:** ordena la salida por fecha de modificación, en orden descendente (el más reciente primero)

Dato curioso: ll

Muchas personas utilizan el atajo **ll** para referirse al comando **ls -lh**, que como se ha explicado, imprime un listado detallado de ficheros y directorios con sufijos para las unidades. Esto se puede configurar con un alias en el archivo de configuración de la shell que se esté usando.

¿Qué muestra el listado detallado?

El primer carácter muestra si es un directorio (d), fichero común (-) o enlace simbólico (l). Seguidamente muestra los permisos con los caracteres r, w, x para el propietario, grupo y otros. Luego aparece el número de enlaces duros que apuntan al mismo i-nodo. Luego aparecen el propietario y el grupo, el tamaño, la fecha y por último el nombre del directorio o fichero.

```
lrwxr-xr-x  1 verdu     staff     6B Feb 13 09:09 enlacePrueba -> prueba
lrwxr-xr-x  1 verdu     staff     9B Feb 13 09:09 enlacePruebaDir -> pruebaDir
-rw-r--r--  1 Autentia  staff     0B Feb 13 09:09 prueba
drwxr-xr-x  2 Autentia  staff    64B Feb 13 09:09 pruebaDir
```

¿Para qué sirven?

En sistemas de tipo Unix, se pueden crear ficheros o directorios de forma sencilla, a través de los comandos **mkdir** (*make directory*) o **touch**. Hacerlo de esta forma es mucho más rápido y eficiente que hacerlo a través de un explorador de archivos.

mkdir

El comando **mkdir** permite crear un directorio nuevo. Si se le indica únicamente el nombre del nuevo directorio, lo creará dentro del directorio actual de trabajo. También se le puede pasar la ruta absoluta del nuevo directorio, o incluso la ruta relativa desde el directorio actual de trabajo. Es importante que todos los directorios intermedios existan porque si no dará un error.

El directorio se creará con permisos de lectura, escritura y ejecución para todos los usuarios, aplicando la máscara del sistema. Por defecto, eliminará los permisos de escritura para “otros”.

Acepta tres parámetros:

- **-v:** conocido como verbose, imprime la información del directorio que se ha creado por pantalla.
- **-p:** crea los directorios intermedios que se le indiquen, en caso de que no existan.
- **-m:** permite modificar los permisos por defecto del nuevo directorio. Por ejemplo, *mkdir -m g=rw- prueba* creará el directorio prueba sin permisos de ejecución para grupos.

Hay que tener en cuenta que para usar mkdir se necesitan permisos de escritura en el directorio actual de trabajo.

touch

El comando **touch** permite crear nuevos archivos y actualizar los tiempos de acceso y modificación de alguno ya existente. Touch admite como parámetro un listado de archivos. Los permisos por defecto de los archivos creados son de lectura y escritura para el propietario y sólo lectura para el resto.

```
~/Autentia touch fichero1 fichero2 && ll
total 0
-rw-r--r--  1 verdu  staff     0B Feb 14 09:03 fichero1
-rw-r--r--  1 verdu  staff     0B Feb 14 09:03 fichero2
```

Touch permite actualizar el tiempo de acceso y modificación de los ficheros a través de parámetros. Los más importantes son:

- **-A:** ajusta los tiempos al valor de tiempo especificado
- **-a:** cambia solo el tiempo de acceso al momento actual
- **-m:** cambia solo el tiempo de modificación al momento actual

¿Para qué sirven?

En sistemas de tipo Unix, se pueden eliminar ficheros y directorios a través de comandos como **rm** o *remove* y **rmdir** o *remove directory*. La ventaja de usar estos comandos es que se pueden usar expresiones regulares para eliminar múltiples archivos de una sola vez.

rm

El comando **rm** permite borrar los ficheros especificados, directorios o enlaces simbólicos. Hay que tener cuidado, ya que por defecto no pide confirmación y puede ser muy destructivo. Por defecto pedirá confirmación para archivos y directorios sin permisos de escritura. Este comportamiento se puede modificar a través de múltiples parámetros:

- **-f:** no pedirá confirmación para ningún fichero, a pesar de que no tenga permisos de escritura.
- **-i:** pedirá confirmación para todos los ficheros que intente eliminar.
- **-I:** pedirá confirmación una sola vez si se eliminan más de 3 ficheros o un directorio recursivamente.
- **-r:** permite borrar un directorio no vacío, eliminando recursivamente su contenido.
- **-v:** indica por pantalla los archivos eliminados a medida que se borran.

rmdir

El comando **rmdir** permite eliminar directorios vacíos. Es el equivalente a utilizar **rm -d**. Con el parámetro **-p** se pueden eliminar todos los directorios padre, si estos están vacíos. Por ejemplo *rmdir -P prueba1/prueba2/prueba3* comenzará eliminando prueba3 si está vacío, luego prueba2 si está vacío y luego prueba1.

Si en algún momento, algún directorio no está vacío, terminará el comando, pero habrá borrado todos los directorios anteriores.

```
 /tmp rmdir -pv prueba1/prueba2/prueba3
prueba1/prueba2/prueba3
rmdir: prueba1/prueba2: Directory not empty
```

En el ejemplo, el directorio *prueba2* no está vacío. Sin embargo, gracias al parámetro -v, muestra que ha borrado el directorio *prueba3*.

¿Para qué sirven?

En ocasiones, queremos copiar un fichero de un directorio a otro o directamente moverlo. Para ello existen los comandos **mv** o *move* y **cp** o *copy* para hacer esta tarea por terminal.

mv

El comando **mv** permite mover archivos de un directorio a otro. El primer argumento que admite es el archivo origen y el segundo el archivo destino. Al moverlo, se le cambiará el nombre al que se le indique como parámetro. Por ejemplo:

mv prueba1 ~/Desktop/test1

Moverá el archivo prueba1 del directorio actual de trabajo al escritorio, cambiándole el nombre a test1.

Entre los parámetros más conocidos que admite están:

- **-v:** el modo verbose indica por pantalla los archivos después de ser movidos
- **-f:** no muestra mensaje de confirmación antes de sobreescribir la ruta destino.
- **-i:** saca un prompt por la salida de error estándar si el movimiento sobreescribe un archivo existente. Si se le indica que continúe se hará efectivo el movimiento.
- **-n:** se indica para que el movimiento no sobreescriba ningún archivo.

cp

El comando **cp** sirve para copiar un fichero de un directorio a otro. Funciona exactamente igual que **mv** en cuanto a argumentos.

Algunos de los parámetros que acepta:

- **-R:** si lo que se copia es un directorio, con este parámetro se copia también el subárbol entero conectado en ese punto.
- **-i:** saca un prompt por la salida de error estándar si el movimiento sobreescribe un archivo existente. Si se le indica que continúe se hará efectivo el movimiento.
- **-l:** crea enlaces duros a los archivos regulares en vez de copiarlos
- **-s:** crea enlaces simbólicos a los archivos regulares en vez de copiarlos
- **-n:** no sobrescribe ningún archivo
- **-v:** modo verbose, para ver por pantalla todo lo que el comando va copiando

¿Para qué sirven?

Muchas veces queremos observar el contenido de un fichero de texto, sin modificarlo, solamente para leer su contenido. Para esta tarea existen tres comandos importantes, **cat**, **head**, y **tail**.

cat

El comando **cat** imprime por pantalla el contenido de un fichero de texto. Se le pueden pasar como argumento múltiples ficheros para que los imprima en orden.

Entre los parámetros más usados están los siguientes:

- **-b:** numera las líneas no vacías, empezando en 1.
- **-n:** numera todas las líneas, empezando en 1.
- **-v:** muestra los caracteres no imprimibles.
- **-e:** parámetro -v y además muestra un $ al final de cada línea.
- **-t:** parámetro -v y además imprime las tabulaciones
- **-s:** si hay varias líneas vacías entre dos textos, las comprime en una sola, dejando un solo espacio entre ambos textos

head y tail

Los comandos **head** y **tail** son comandos que sirven para examinar el principio o el final de un fichero respectivamente. Si no se le añaden parámetros, muestran las 10 primeras líneas o las 10 últimas del fichero que se les pase como argumento.

Ambos comparten dos parámetros importantes:

- **-n:** este parámetro permite indicar el número de líneas a mostrar
- **-c:** número de bytes a mostrar

Adicionalmente, **tail** tiene algunos más:

- **-b:** número de bloques a mostrar. Cada bloque tiene 512B.
- **-r:** invierte el orden en el que se muestran las líneas.
- **-f:** este parámetro hace que tail quede a la espera de datos adicionales añadidos al fichero. Su uso es muy común para monitorizar logs en tiempo real.

¿Para qué sirven?

Si alguna vez nos encontramos con algún fichero de muchas líneas o un input muy grande quizás nos interese procesarlo y filtrarlo. Aquí entran en juego los comandos **grep** y **awk**.

grep

El comando **grep** permite seleccionar las líneas de una entrada, ya sea de un fichero o de la salida de otro comando, que coincidan con uno o varios patrones.

```
~/Desktop ls | grep -n -e ^pr -e Dir$
8:enlacePruebaDir
10:prueba
11:pruebaDir
```

Grep acepta muchos parámetros, pero los más comunes son:

- **-c:** en lugar de mostrar las líneas coincidentes muestra el número total
- **-e:** después de este parámetro se especifica un patrón. Puede haber varios patrones en un mismo grep.
- **-r:** busca recursivamente dentro de todos los subdirectorios del directorio de trabajo
- **-v:** muestra las líneas que no coinciden con el patrón.
- **-i:** ignora la distinción entre mayúsculas y minúsculas
- **-n:** numera las líneas de salida
- **-E:** nos permite usar expresiones regulares complejas.
- **-o:** sólo muestra la parte de la línea que coincide con el patrón
- **-f:** extrae los patrones del archivo que se le especifique

awk

El comando **awk** se utiliza para filtrar ficheros tratando cada línea, para por ejemplo mostrar cierta información de la misma. Por ejemplo:

```
~/Desktop ll | awk '{print $9 " owned by " $3}'
 owned by
Verdu owned by verdu
comandos.txt owned by verdu
```

Imprime el nombre del fichero y a quién le pertenece en el directorio actual de trabajo. Esto es porque trata a las líneas por columnas, y con el $ se le indica el número de columna. Por defecto, el separador de columnas es el espacio, pero se puede cambiar.

La estructura del comando consta de dos partes, la **condición** y el **comando.**

Por ejemplo, el comando anterior imprimiría una primera línea con el texto owned by. Esto es porque **ll** saca una línea adicional al principio. Esto se puede eliminar poniendo como condición que solo imprima a partir de la línea 2. (NR>1)

¿Para qué sirven?

Muchas veces, pueden acumularse procesos en segundo plano que querríamos cancelar, o ver su estado. Esto se puede hacer desde una terminal con los comandos **ps** para ver los procesos activos y **kill** para eliminarlos.

ps

El comando **ps** viene de *process status* y como su nombre indica, con él podremos ver la lista de procesos activos de una terminal concreta o de todo el sistema, dependiendo de los parámetros. Este comando nos muestra el PID de los procesos, entre muchas otras cosas. El PID es el identificador único del proceso y con él podemos controlarlo.

```
  PID TTY           TIME CMD
28492 ttys000    0:00.00 sleep 100
```

Admite los siguientes parámetros:

- **-A:** muestra los procesos de todos los usuarios
- **-a:** muestra los procesos de una tty determinada
- **-T:** muestra los procesos de la terminal actual
- **-f:** muestra los parámetros que se le pasaron al proceso
- **-U:** muestra los procesos del usuario seleccionado, pasándole el ID
- **-p:** muestra los procesos que cumplan con un PID determinado

kill

El comando **kill** sirve para terminar un proceso. Este comando manda una señal al proceso con el PID especificado y dependiendo de ésta, el proceso se comportará de formas diferentes.

Como parámetros se le pueden pasar dos:

- **-s:** indica el nombre de la señal que se le quiere enviar al proceso. Por ejemplo, INT.
- **-sig_number:** se le indica el número correspondiente a la señal que se le quiere enviar. Por ejemplo
  - -2: Interrupción
  - -3: Quit
  - -6: Abortar
  - -9: Matar
  - -15: Terminar (por defecto)

```
 ~/Autentia  kill -2 28492
[1]  + 28492 interrupt  sleep 100
```

¿Cómo funcionan?

En UNIX, los permisos se gestionan mediante un esquema de tres tipos de permiso: **lectura, escritura** y **ejecución**. Estos permisos se asignan de forma diferente para el propietario del fichero o directorio, el grupo propietario y otros usuarios.

¿Qué significan los permisos?

Con el comando **ls -l** se pueden ver los permisos de todos los ficheros y directorios del directorio actual de trabajo. En ellos, veremos repetido el esquema **rwx** tres veces. Los tres primeros muestran los permisos del propietario, los tres siguientes los del grupo propietario y los tres últimos los permisos de otros usuarios.

```
drwxr-xr-x  2 Autentia  staff    64B Feb 13 09:09 pruebaDir
```

Cuando aparece un guión quiere decir que se carece de ese permiso.
Dependiendo de si es un fichero o un directorio los permisos significan una cosa u otra:

- **Lectura (r):** sobre un archivo quiere decir que se puede abrir y leer el contenido, mientras que sobre un directorio quiere decir que se puede listar el contenido del mismo, por ejemplo, mediante el comando ls.
- **Escritura (w):** sobre un archivo quiere decir que se puede modificar su contenido, borrarlo o cambiarle los permisos, mientras que sobre un directorio quiere decir que se puede crear o eliminar ficheros dentro de él.
- **Ejecución o x:** sobre un archivo quiere decir que se puede ejecutar, mientras que sobre un directorio quiere decir que puede entrar o navegar a través de él, por ejemplo con el comando cd.

chmod

El comando **chmod** nos permite cambiar los permisos a un fichero o directorio. Existen dos formas:
Los permisos en octal se representan con números del 0 al 7. Hay una relación entre el número binario que representa y los permisos que se atribuyen:

r	w	x
1	0	0

Por ejemplo, para dar todos los permisos al propietario y grupo del fichero y a otros sólo permiso de lectura:

- *chmod 774 fichero*

La otra opción es usar la **sintaxis propia de chmod,** la cual se basa en abreviaturas. Por ejemplo:

- *chmod ug=rw* indica que al propietario y al grupo se le otorgan permisos de lectura y escritura.
- *chmod o+x* añade permisos de ejecución a otros
- *chmod go-w* elimina permiso de escritura al grupo y a otros

El comando **chown** permite cambiar el propietario y grupo a un fichero o directorio: *chown autentia:staff fichero*.

¿Qué son?

Las tuberías son herramientas que unen la salida estándar de un comando con la entrada de otro comando mientras que las redirecciones permiten trasladar información de un tipo a otro, por ejemplo desviar la salida estándar.

Tuberías

Las tuberías permiten enlazar comandos, evitando así depender de archivos intermedios. Se representan con el símbolo **|** y se escriben entre dos comandos. Por ejemplo:

```
~/Desktop ll | head -n 3 | grep txt | awk '{print $9}'
comandos.txt
```

En el ejemplo se ve cómo se enlazan 4 comandos diferentes, que cogen la salida del anterior y la utilizan como su entrada. En este caso se hace un listado detallado de lo que hay en el directorio actual de trabajo, se leen solamente las 3 primeras líneas, se filtra por txt y se muestra únicamente el nombre de los archivos coincidentes.

En caso de que en el medio del proceso se quiera guardar algún dato sin alterar el flujo de tuberías, podemos hacerlo con el comando **tee**. Por ejemplo, guardar en un fichero el resultado entre el comando *head* y el comando *grep*:

```
~/Desktop ll | head -n 3 | tee zzz | grep txt | awk '{print $9}'
comandos.txt

~/Desktop cat zzz
total 8
drwxr-xr-x  7 verdu  staff   224B Feb  9 09:29 Verdu
-rw-r--r--@ 1 verdu  staff   219B Feb  8 11:40 comandos.txt
```

Redirecciones

En UNIX existen tres dispositivos de entrada/salida fundamentales:

- **Entrada estándar:** descriptor de archivo 00 y por defecto el teclado.
- **Salida estándar:** descriptor de archivo 1 y por defecto la pantalla de la terminal.
- **Error estándar:** descriptor de archivo 2 y por defecto la pantalla de la terminal.

Estos dispositivos por defecto de entrada/salida se pueden alterar gracias a las redirecciones, las cuales se representan con el símbolo **>** o **<**.

- *comando < entrada.txt*: el símbolo **<** cambia la entrada estándar del comando, en este caso a un fichero txt.
- *comando > salida.txt*: el símbolo **>** cambia la salida del comando al archivo especificado. Es equivalente a escribir **1>**. Si se especifican dos símbolos **>>** no se sobreescribe el contenido del fichero salida.txt.
- *comando 2> errorLog.txt*: el símbolo **2>** redirige los errores del comando al fichero especificado.

Enlaces y el comando ln

autentia

¿Qué son?

Los enlaces en sistemas tipo Unix son punteros que apuntan a ficheros o directorios. Hay dos tipos de enlaces, los **enlaces duros** y los **enlaces blandos** o **simbólicos.** Cada uno de ellos tiene características diferentes. El comando **ln** permite crear estos enlaces.

Enlaces duros

Los enlaces duros apuntan a un **nodo-i**. Los **inodos** o **nodos-i** son una estructura de datos propia de sistemas tipo Unix, y almacenan toda la información sobre un fichero o directorio.

Un enlace duro asocia dos o más **ficheros** al mismo nodo-i.. Eliminar uno de estos ficheros no eliminará el nodo-i, a no ser que sea el último que hacía referencia a ese nodo. Se puede ver el nodo-i y el número de enlaces duros de ese nodo con **ls -li**.

Este tipo de enlaces también se denominan **enlaces físicos** ya que hay dos o más ficheros físicos apuntando al mismo nodo. No se pueden crear enlaces duros entre directorios ni entre sistemas de ficheros distintos. Si se modifica el contenido de un enlace se ve reflejado en el resto.

Para crear un enlace duro, se puede usar el comando **ln.**

```
~/Desktop/PruebaInodos ls -li
total 8
1730279 -rw-r--r--  1 verdu  staff  15 Mar  3 13:55 original

~/Desktop/PruebaInodos ln original duro

~/Desktop/PruebaInodos ls -li
total 16
1730279 -rw-r--r--  2 verdu  staff  15 Mar  3 13:55 duro
1730279 -rw-r--r--  2 verdu  staff  15 Mar  3 13:55 original

~/Desktop/PruebaInodos echo "Leo esto desde el duro" > duro

~/Desktop/PruebaInodos cat original
Leo esto desde el duro
```

Enlaces simbólicos

Los enlaces simbólicos o enlaces blandos permiten el acceso a un **directorio** o **fichero** situado físicamente en una localización distinta, es decir, apuntan al fichero original. Si se modifica alguno de estos enlaces, en realidad se está modificando el fichero o directorio original. Esto se puede comprobar con el comando **pwd -P**, que muestra la ruta física en la que nos encontramos. Estos enlaces se pueden hacer entre sistemas de ficheros distintos.

En este caso, si se elimina el fichero original, todos los enlaces que apuntan a él se romperán y darán un error.

Se pueden crear enlaces blandos con el comando **ln -s**.

```
~/Desktop/PruebaBlandos ls -li
total 0
1732150 drwxr-xr-x  2 verdu  staff  64 Mar  3 14:21 original

~/Desktop/PruebaBlandos ln -s original copia

~/Desktop/PruebaBlandos ls -li
total 0
1732166 lrwxr-xr-x  1 verdu  staff   8 Mar  3 14:21 copia -> original
1732150 drwxr-xr-x  2 verdu  staff  64 Mar  3 14:21 original

~/Desktop/PruebaBlandos cd copia

~/Desktop/PruebaBlandos/copia pwd -P
/Users/verdu/Desktop/PruebaBlandos/original

~/Desktop/PruebaBlandos/copia touch hola

~/Desktop/PruebaBlandos/copia ls ../original
hola
```